Sichere Entwicklung von mobilen Applikationen

SimoBIT-Workshop “Mobile IT-Sicherheit”, 07.06.2010

Dr. Karsten Sohr,
Technologie-Zentrum Informatik der Universität Bremen

Inhalt

- Motivation
- Die Methoden des Angreifers
- Software-Sicherheit vom Standpunkt des Anwenders
- Software-Sicherheit vom Standpunkt des Entwicklers
- Forschungsthema im Bereich „Sicherheit mobiler Applikationen“
- Zusammenfassung und Ausblick

Warum Software-Sicherheit? – Drei Trends

1. **Zunehmende Vernetzung**
   - Export von Legacy-Anwendungen mit geringer Sicherheit durch Service-Oriented Architectures
   - Mega-Lösungen wie z.B. SAP, Oracle, PeopleSoft
   - Software-Rahmenwerke: JEE, .Net, WebSphere
2. **Erweiterbarkeit**
   - Nachladen neuer Software
   - Beispiel: Plug-in von Browsern
   - In Zukunft: Vermehrt Nachladen von Applikationen für Mobiltelefone/Smartphones

Warum Software-Sicherheit? – Drei Trends

3. **Komplexität**

   Immer unfangreichere Software

   - Komplexe Betriebssysteme
   - Identitätsmanagement-Systeme mit Provisioning
   - XML und XML-Parser
   - Model View Controller-Paradigma (z.B. Struts)
   - Application Server, Web Container (z.B. JBoss)
   - Datenbanken (Oracle)

   „Hello World“-Programm in IBM Websphere: 2 MB !?

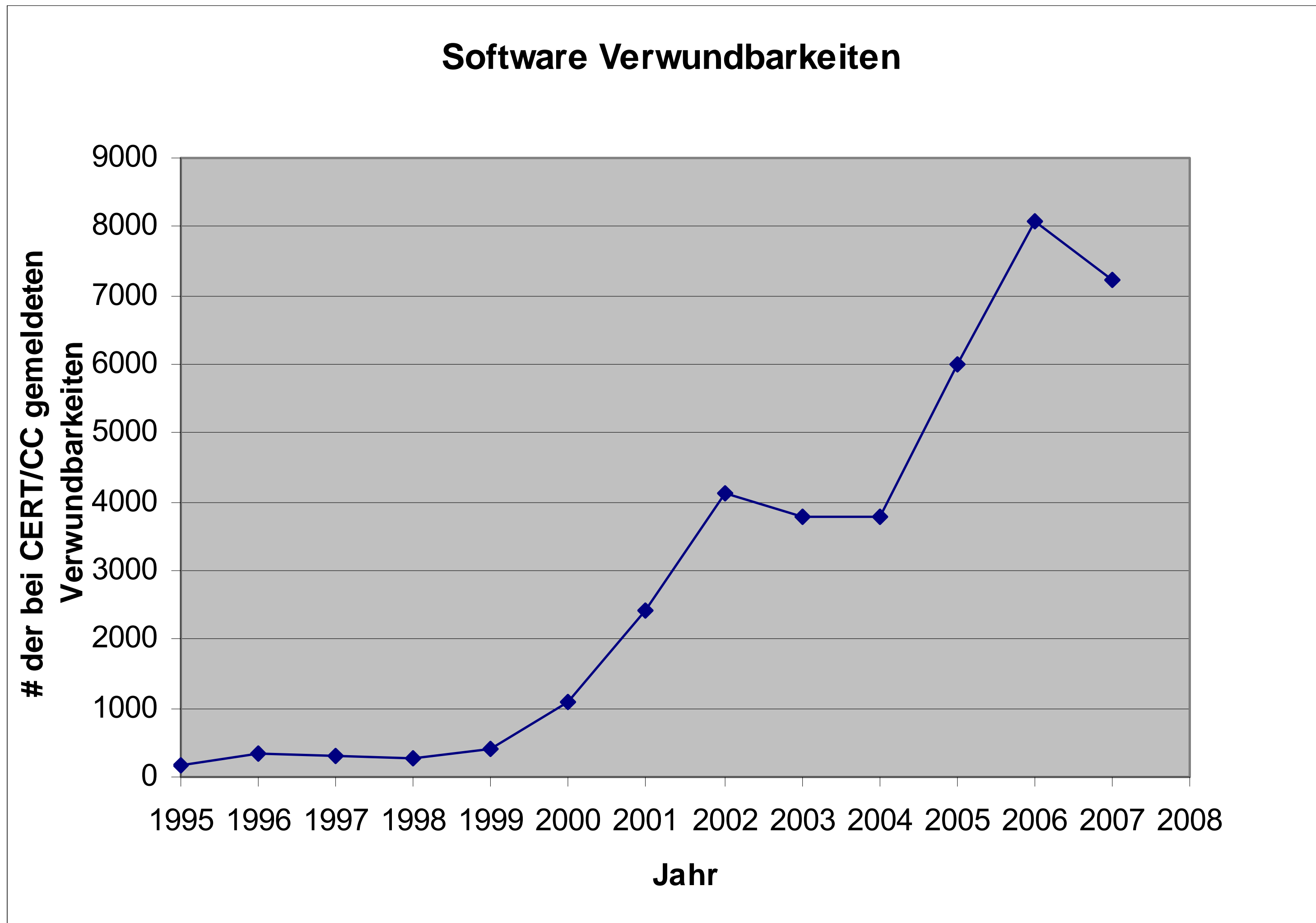
Wachsende Komplexität: Beispiel Windows

- Win 3.1 (1990): ca. 3 Millionen Zeilen Code
- Win 95 (1997): ca. 15 Millionen Zeilen Code
- Win 2K (2001): ca. 35 Millionen Zeilen Code
- Win XP (2001): ca. 40 Millionen Zeilen Code

Konsequenzen aus den drei Trends

- Komplexe und umfangreiche Software enthält Fehler
- „Mehr Zeilen, mehr Fehler“:
  Richtwert: 5-50 Fehler pro 1000 Zeilen Code! (→ 200.000 Fehler in Win XP?)
- Vernetzung bietet die Möglichkeit, solche Schwachstellen entfernt auszunutzen (entferntes Aufrufen von Software)
- Software-Verwundbarkeiten: Ursache für viele Sicherheitsprobleme
- Wachsende Anzahl an Software-Verwundbarkeiten

- Angreifer nutzen allerdings *bislang nur* wenige Arten von Fehlern
  - Niedrig hängende Früchte
  - Bleibt das so?


Software Verwundbarkeiten
der bei CERT/CC gemeldeten Verwundbarkeiten
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008
Jahr

Die Methoden der Angreifer

- Buffer-Overflows
- SQL-Injection
- Cross-Site-Scripting
- Race Conditions
- Privilege Escalation

Buffer-Overflows

- Häufigste Einbruchmethode in Server (insbesondere in Web-Server)
- Altbekanntes Problem (schon in den 60er Jahren bekannt)
- „Attack of the decade“ (Bill Gates)
- Die meisten Viren/Würmer nutzten Buffer-Overflows aus (Morris-Wurm, Code Red, Blaster, Conficker)
- Ziel: Einschleusen von Code
- Ausnutzen von Programmierfehlern

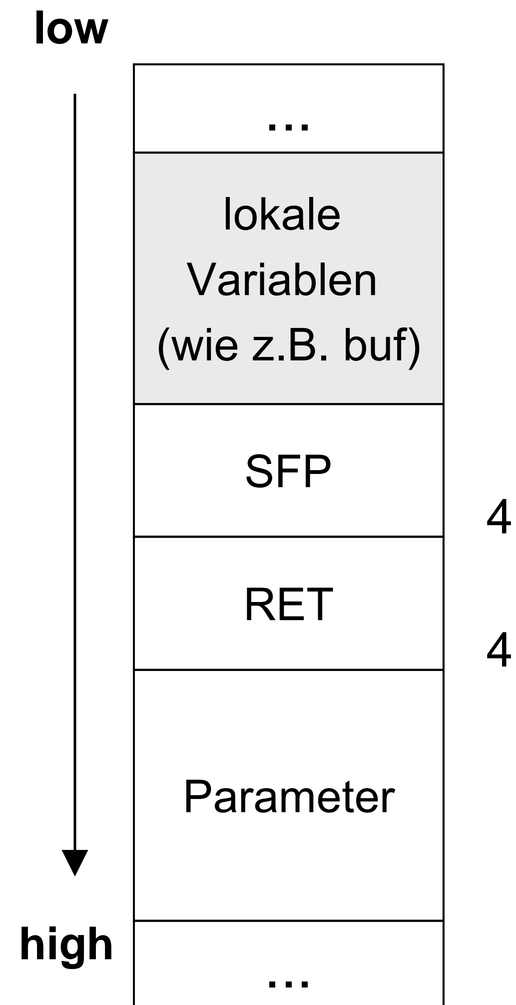
Nicht korrekt geprüfte Eingaben

```
char buf[42];
gets(buf);
```

- **Problem**:
  - C-Funktion `gets` überprüft nicht die die Länge der Eingabe („unchecked buffer“)
  - Ist die Länge der Eingabe größer als 41 (Nullterminierung beachten!), dann wird umgebender Speicher überschrieben.
  - Lokale Variablen sind auf dem Call-Stack

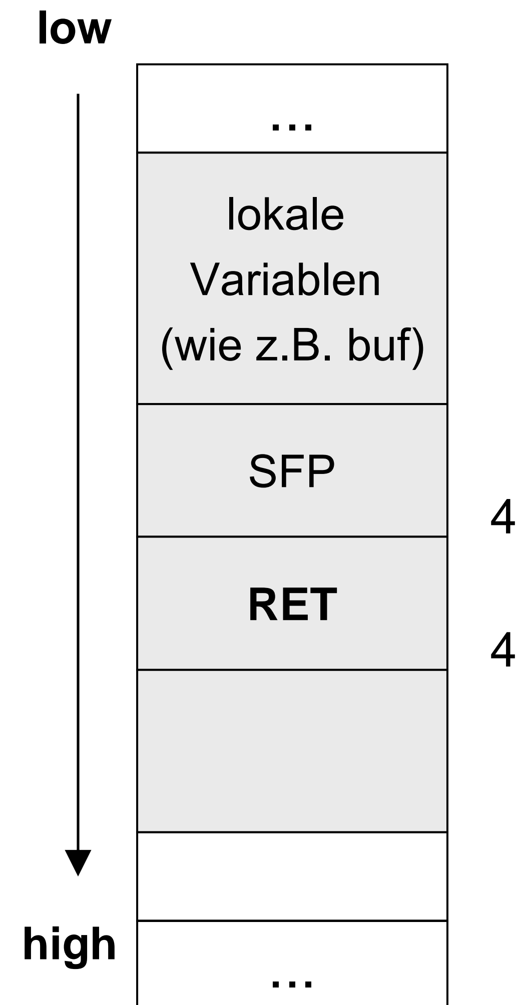
Call-Stack

- Lokale Variablen von Unterprogrammen werden auf Call-Stack abgelegt
- **Beispiel**: Stack für einen Unterprogrammaufruf bei der x86-Architektur
- RET: Rücksprungadresse
- SFP: Stack Frame Pointer



Überschreiben der Rücksprungadresse

- RET kann durch entsprechend präparierten Eingabestring so manipuliert werden, dass Rücksprung in den Exploit-Code führt
- Exploit-Code wird in Stack untergebracht
- Beispiel für Exploit-Code unter Linux/Unix: `execl` zur Ausführung einer Shell auf angegriffenem System
- Nützlich, wenn das Programm mit der Buffer-Overflow-Verwundbarkeit mit Admin-Rechten läuft



Beispiel: Conficker

- **Schwachstelle: CVE-2008-4250 - Buffer Overflow im RPC Interface des Windows Server Service**

  Der Windows Server Service ist für die Freigabe von Shares, Druckern und Named Pipes im Netz zuständig. Im RPC Interface des Dienstes lässt sich ein Buffer Overflow auslösen. Ein Angreifer kann diese Schwachstelle über das Netz dazu ausnutzen, beliebigen Code mit SYSTEM Rechten auszuführen.

  Infizierung möglich bei offenen Ports 139/445 und ungepatchtem Windows (kein E-Mail-Wurm!)

Fazit: Buffer-Overflows

- **Haupt-Ursache für Buffer-Overflows**: Programmierfehler:
  Verwendung von vordefinierten Bibliotheksroutinen in Programmiersprachen wie C oder C++ ohne Bereichsprüfung:
  - strcpy(), strcat(), gets() in C
  - Es gibt Alternativen strncpy(), strncat(), fgets().
- **Bemerkung**:
  Die meisten BS-Dienste (Unix, Windows) sind in C, C++ programmiert.
  Fast alle Würmer nutzen Buffer-Overflows aus

SQL-Injection

- Viele Web-Anwendungen (z.B. PHP) speichern Kundendaten, Login-Informationen in Datenbanken
- Die Daten werden von den Benutzern in Eingabefelder eingegeben wie z.B. E-Mail-Adressen
- PHP bietet z.B. eine bequem benutzbare Datenbankschnittstelle, um die eingegebenen Daten direkt in die Datenbank auf dem Web-Server zu schreiben
- Wie bei den Buffer-Overflows: Angreifer definiert Daten!

... Angreifer gibt die Daten ein

Beispiel:

Eingabefeld für eine E-Mail-Adresse in einem Web-Formular für Kunden der Firma „Unsicher“

Angenommen, PHP-Anwendung nutzt SQL-Statement:

```
SELECT email, passwd, login_id, name
FROM members
WHERE email='Daten vom Netz'
```

unsere Eingabe = `bla@tzi.de'; DROP TABLE members;--`

Im SQL-Statement:

```
...WHERE email ='bla@tzi.de'; DROP TABLE members;--'
```

Hilfe für einen Angreifer: sprechende Fehlermeldungen

```
ERROR open DbFetch: 1054: Unknown column '404e'
  in 'where clause,

ERROR_TEIACourseDescriptionERROR_TEIAOpenCsvFile

Warning: Invalid argument supplied for foreach()
  in
/home/apliq/teia/htdocs/template/course_detail_vi
  ew.inc.php on line 76
```

Cross-Site Scripting (XSS)

- Webseiten können Skripte enthalten (z.B JavaScript)
  - Werden auf Kunden-Browser ausgeführt
  - Haben Zugriff auf Cookies der Website
- Webseiten können benutzerdefinierte Daten enthalten
  - Vorherige Eingaben eines anderen Benutzers
  - Evtl. auch unerwartete Reaktion auf URL-Parameter
- Angriff: Angreifer unterschiebt Opfer ein Skript

Durchführung eines Angriffs

- **Beispiel**:
  http://auction.example.com/filename.html liefert eine Webseite zurück mit der Meldung
  `404 page does not exist: filename.html.`
- Angreifer schickt dem Opfer nun einen präparierten Link (z.B. per E-Mail):
  http://auction.example.com/<script>alert('hello')</script>
- Beim Abruf des Links wird das Skript zurückgeliefert und aufgerufen (und zwar nicht auf dem Webserver, sondern auf dem Rechner des Opfers)

Fazit: XSS-Angriffe

- Angreifer kann auf Opferrechner beliebige Javascript-Befehle ausführen:
  - Cookie auslesen und woanders ablegen
  - Eingabefenster für Passwörter simulieren
- Session-Übernahme möglich durch Stehlen eines Session-Cookies (im Auktionsbeispiel evtl. besonders kritisch)
- Angegriffener: primär der Nutzer, aber z.B. durch Rufschädigung auch der Betreiber der Website
- Betreiber der Website sollte solche Fehler beseitigen
- Schutz: Validation von Eingaben (durch den Betreiber der Web-Site)

TOCTOU-Angriffe

- TOCTOU: Time of Check, Time of Use
- Anderer Begriff: Race Conditions (Wettlaufsituationen)
- Basiert auf Nebenläufigkeit in Betriebssystemen und Anwendungen
- Operationen oft nicht atomisch
- Zugriffskontrolle kann ausgehebelt werden
- Häufiges Muster für Verwundbarkeit:
  1. Überprüfung der Zugriffsrechte (Time of Check)
  2. Durchführung der sicherheitskritischen Operation (Time of Use)
- TOCTOU-Angriffe: Es kommt für den Angreifer auf den richtigen Zeitpunkt an
  - Ausprobieren, Angriff evtl. oft wiederholen

Privilege Escalation

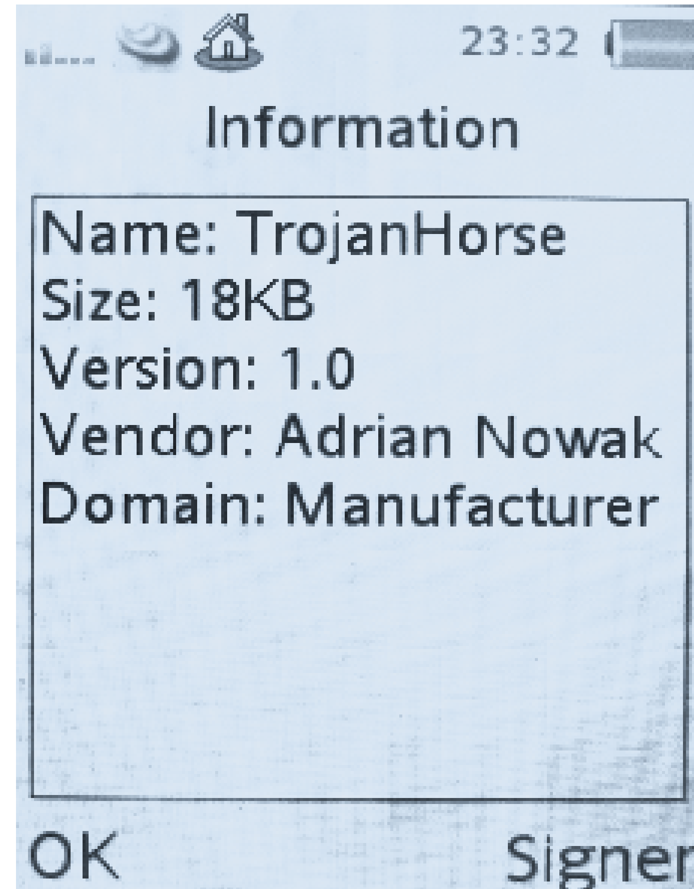
- Ziel: Erhöhte Zugriffsberechtigungen auf dem angegriffenen System
- Beispiel: Admin-Rechte unter Windows, root unter Unix/Linux
- Oft Ausnutzen von Buffer-Overflows, Race Conditions

- Geht auch bei Mobiltelefonen (→ nächste Folie)

Angreifen von Java-Handys

- Sicherheitslücke in Sony-Ericsson-Handys (2007, Adrian Nowak)
  - Ca. 100 Millionen Handys verwundbar
- Internes Dateisystem, auf das normale Benutzer nicht direkt zugreifen dürfen
- Link-Dateien ungeschützt, dürfen aber nicht einfach über Java-Programm erzeugt werden (`write, create` verbieten das)
- Aber: Fehlerhafterweise erlaubt `rename`-Funktion doch das Erzeugen von Links
- Möglichkeit, jede interne Datei zu überschreiben:
  - Auch Hersteller-Zertifikate
  - Benutzer wird Hersteller (oder Netzbetreiber?)

Angreifer wird Hersteller…



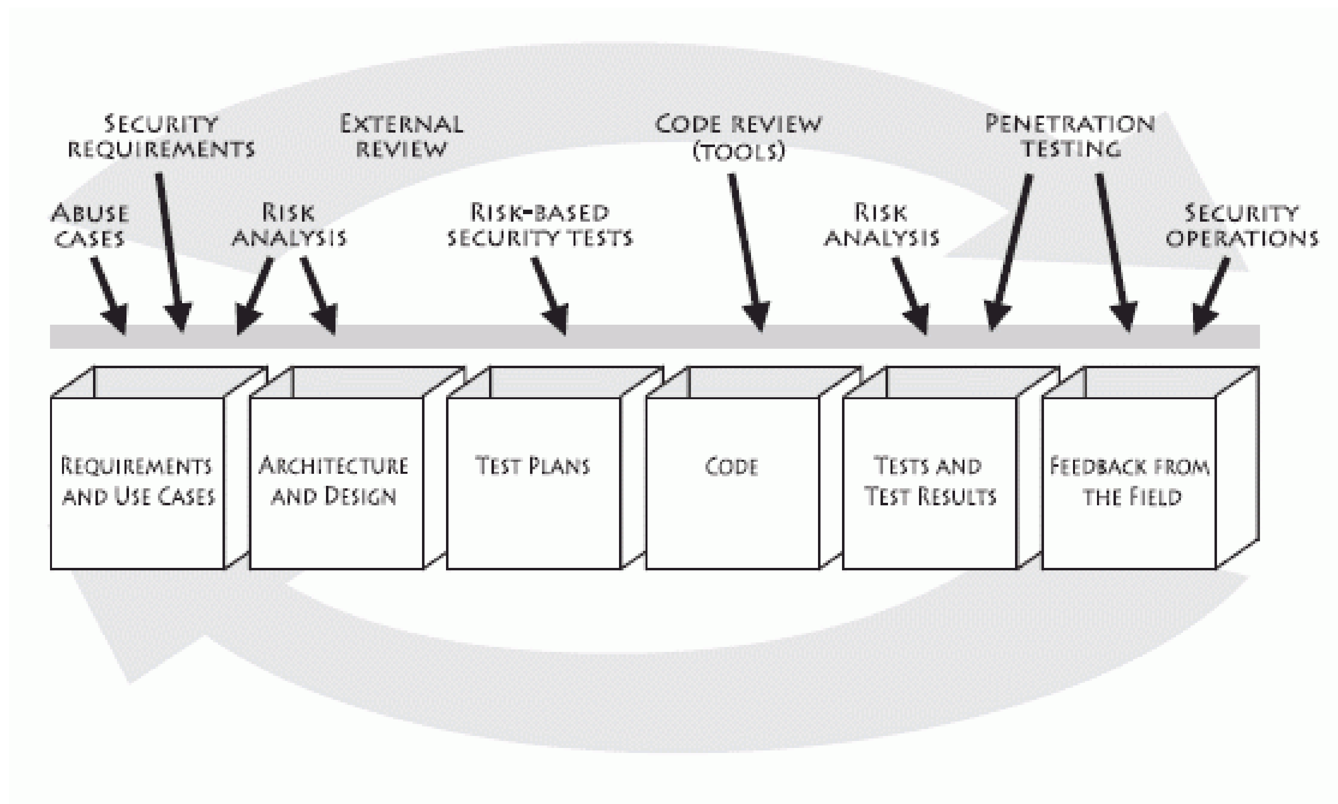
Software-Sicherheit vom Standpunkt des Anwenders

- Vorsicht beim Exportieren interner Software
- Sicherheitsrisiken der Applikation kennen
- Regelmäßiges Lesen von Bugtraq, Cert/CC-Advisories, BSI-Webseite über SW von Drittanbietern
- Patchen von Software
  - Nicht nur OS, sondern auch SAP, Datenbanken, Adobe, Bildverarbeitung, …
  - Sicherheitslöcher durch vermeintlich harmlose Anwendungen
- Penetrationstests der Applikationen durchführen (lassen)
  - Einsatzumgebung testen (z.B. Firewall-, Application-Server -Konfiguration)
- Binärcode-Analyse

Software-Sicherheit vom Standpunkt des Entwicklers

- Software-Sicherheit ≠ Sicherheitssoftware
- Sicherheits-Features: SSL, Passwort-Mechanismen, Zugriffskontrolle, Digitale Signaturen, …
  - Sicherheitslücken aber oft in „harmlosen“ Applikationen (z.B. iTunes)
- Software-Sicherheit in Entwicklungsprozess integrieren
  - Pen-Test am Ende wichtig
    - Aber zu spät und zu geringe Abdeckung
    - Durchführung oft von Netzsicherheitsexperten
  - Hohe Kosten bei Design-Fehlern, schwer korrigierbar
  - Sicherheitsexperte McGraw: „Seven Touchpoints“ of Software Security

Sieben Schritte der sicheren SW-Entwicklung



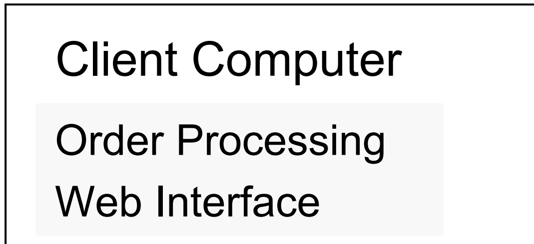
Wichtige Aspekte: Risikoanalyse der SW-Architektur

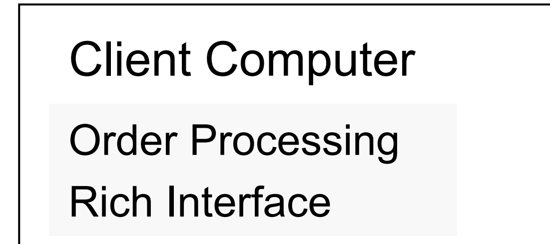
- Designfehler 50% aller Sicherheitsprobleme von Software
- Beispiele
  - Rollenbasierte Zugriffskontrolle inkonsistent über mehrere Schichten einer Multi-Tier-Anwendung
  - Falscher Einsatz von Krypto (Schlüssellänge zu klein), WEP als Beispiel
  - Fehlerhafte Vertrauensbeziehungen von Applikationen/Modulen (später)

Wichtige Aspekte: Risikoanalyse der SW-Architektur

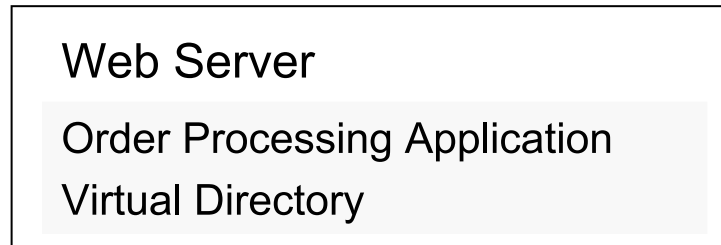
- Vorgehensweise
  - Erstellen einer Abbildung mit den Hauptkomponenten des Systems (Forest Level-View)
  - Durchsuchen der Literatur nach bekannten Sicherheitsproblemen
    - Those who cannot remember the past are *condemned to repeat* it (George Santayana)
  - Identifizierung von Sicherheitsproblemen verwendeter SW-Rahmenwerke
  - Zusammensetzen in einer Gruppe, um unbekannte Risiken zu identifizieren
    - Parallelisierung
    - Forest-Level-Abbildung hilfreich
  - Ausnutzen eines Werkzeuges wie z.B. beim STRIDE-Modell von Microsoft

Client Tier

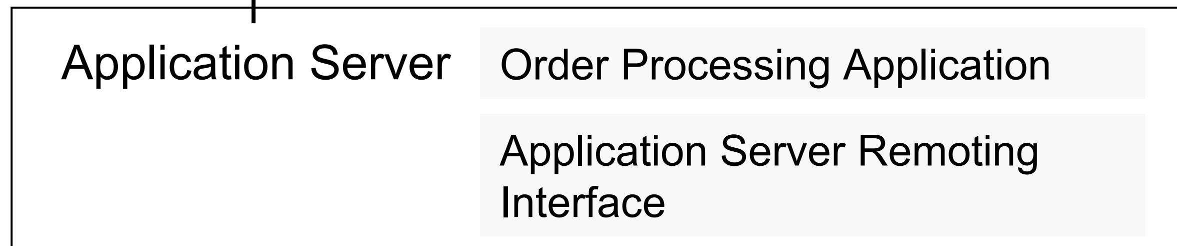




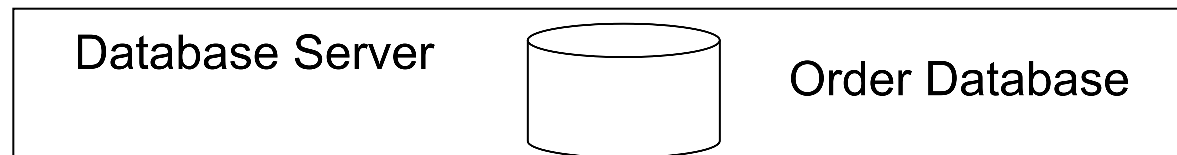
Web Tier



Application Tier



Data Tier



Wichtige Aspekte: Code Review mit einem Werkzeug /1

- Statische Analyse von SW
- Kann viele (aber nicht alle) Implementierungsfehler wie z.B. Buffer-Overflows, XSS-Probleme, SQL-Injection, … finden
- Gut, aber nicht perfekt
- Keine Architekturfehler
- Nicht manuell, sondern werkzeuggestützt
- Integration verschiedener Regeln von SW-Fehlern (Taxonomie von gängigen Implementierungsfehlern)

Wichtige Aspekte: Code Review mit einem Werkzeug /2

- Einige akademische Werkzeuge
  - MOPS, BOON, Eau Claire
- Aber auch kommerzielle Werkzeuge wie die Fortify Source Code Analysis Suite
  - Unterstützt u.a. C/C++, C#, Java, JSP, XML, SQL
  - http://www.fortify.com/
  - GUI nächste Folie
  - Weitere Werkzeuge später

Fortify Audit Workbench - webgoat - [C:\Programme\Fortify Software\SCAS-Demo4.0.0\Tutorial\java\audits\webgoat\webgoat.fpr] *

File Edit Tools Options Help

Issues - Broad (Fortify Default)

Hot (68) | Warning (6) | Info (5) | All (79)

Group by: Category

- Race Condition: Singleton Member Field (Stru
  - HammerHead.java:86 (Singleton Membe
  - LessonSource.java:53 (Singleton Member
- SQL Injection (Data Flow) - [10 / 10]
  - BlindSqlInjection.java:76 (SQL Injection)
  - ChallengeScreen.java:183 (SQL Injection)
  - DOS_Login.java:97 (SQL Injection)
  - DOS_Login.java:111 (SQL Injection)
  - SqlNumericInjection.java:83 (SQL Injectic
  - SqlStringInjection.java:78 (SQL Injection)
  - StoredXss.java:70 (SQL Injection)
  - ThreadSafetyProblem.java:71 (SQL Injec
  - ViewDatabase.java:59 (SQL Injection)
  - WsSqlInjection.java:191 (SQL Injection)
- Cross-Site Scripting (Data Flow) - [54 / 54]
- Password Management: Hardcoded Passwor

LessonSource.java | BlindSqlInjection.java

```
69
70
71
72
73              } else {
74
75                  Statement statement = connection.createStatement( ResultSet.TYPE_SCROLL_INSENSITIVE,
76                  ResultSet results = statement.executeQuery( query );
77
78                  if ( ( results != null ) && ( results.first() == true ) )
79                  {
80
...
90
91          catch ( Exception e )
92          {
93              s.setMessage( "Error generating " + this.getClass().getName() );
94              e.printStackTrace();
95          }
96
97          return ( ec );
98
99
100
101
102
103
104
```

Analysis Trace

- getParameterValues(return) - ParameterPars
- [assignment to values] - ParameterParser.ja
- [return values] - ParameterParser.java:601
- getRawParameter(return) - ParameterParser
- return - ParameterParser.java:572
- getRawParameter(return) - BlindSqlInjection.
- [assignment to accountNumber] - BlindSqlInj
- [assignment to query] - BlindSqlInjection.java
- executeQuery(0) - BlindSqlInjection.java:76

Summary | Details

Location: src\lessons\BlindSqlInjection.java:76

Analysis: Exploitable — Impact: Medium

Status: Reviewed — List: Hot

Comments: Reading a value from an HTTP request and including it in a dynamic SQL query is a sure-fire path to SQL injection.

File Bug... | Suppress issue

SQL Injection (Input Validation and Representation, Data Flow)

Constructing a dynamic SQL statement with user input may allow an attacker to modify the statement's meaning or

View More Details

Weitere kommerzielle Werkzeuge für Code Review

- Werkzeuge von verschiedenen Unternehmen wie z.B.
  - Coverity
  - Klocwork
  - Ounce Labs
  - Parasoft
  - Veracode
  - HP, IBM, Microsoft

- Gartner-Studie zu diesen Werkzeugen:
  Magic Quadrant for Static Application Security Testing
  (www.fortify.com/magicquadrant/)

Analyse des Binär-Codes

- Was ist, wenn kein Quelltext vorhanden?
  - Häufige Situation für Unternehmen
  - Analyse des Binärcodes als Lösung dieses Problems?
- Für Java, C# relativ einfach (Dekompilierung von Bytecode)
  - Unterstützung der meisten Code Review-Werkzeuge wie z.B. Fortify
- Für C schon schwieriger
  - Veracode (allerdings kein Vergleich mit anderen Toolanbietern möglich)

Wichtige Aspekte: Software Penetrationstests

- Integration in SW-Entwicklungsprozess
  - Ergebnisse der Risikoanalyse nutzen
  - Ergebnisse der Pen-Tests in den SW-Entwicklungszyklus einbringen;
  - nicht nur das offensichtliche Sicherheitsproblem beheben:
    Declare victory and go home

- Nicht als erste Sicherheitsüberprüfung im SW-Entwicklungsprozess, letzter Test vor dem Deployment!

- Werkzeuge nutzen

Software Penetrationstests – Werkzeuge nutzen

- Disassembler (z.B. IDA, www.hex-rays.com/idapro/), Debugger, APISPY32 (download.softwareload.de/APISpy32/64)
- http Analyzer (www.ieinspector.com/httpanalyzer),
- Fuzzer (www.infosecinstitute.com/blog/2005/12/fuzzers-ultimate-list.html) wie z.B. Spike:
  - Zufällig fehlerhafte Daten erzeugen und dann (Web-)Applikation damit testen
  - BMBF-Projekt *SoftSCheck* zum Testen von Fuzzern (FH Rhein-Sieg, MS)
- SPI Dynamics (www.spidynamics.com), jetzt HP

Forschungsthema: Sicherheitsanalyse von Android-Applikationen /1

- Problem:
  - Erweiterbarkeit durch immer neue Applikationen
  - Applikationen für sich betrachtet sicher, aber im Zusammenspiel unsicher
  - Beispiel: **App A** liest Location-Daten, **App B** darf Daten ins Internet schreiben
  - Nutzer bestätigt meist ohne Kontrolle die von einer App angeforderten Berechtigungen
- Analyse des Zusammenspiels von Android-Applikationen

Forschungsthema: Sicherheitsanalyse von Android-Applikationen /2

- Lösungsansatz:
  - Nutzen eines Reverse Engineering-Werkzeugs (Bauhaus), um Software-Architektur der zusammengesetzten Applikation zu analysieren (High-Level-Sicht)
  - Grobauswahl der relevanten Klassen, Methoden
  - Anschließend Analyse auf der Code-Ebene (auch Bytecode)

- Analyse (von Teilen) des Android-Markets
- Analyse der auf einem Endgerät installierten Software

Zusammenfassung & Ausblick

- Komplexere Software, mobiler Code und Vernetzung führen zu mehr Sicherheitsproblemen
- Bislang vor allem Buffer-Overflows, in Zukunft aber auch andere Probleme
- Pentests nicht ausreichend, Integration von Sicherheit in den SW-Entwicklungsprozess erforderlich
- Code Review mit Werkzeug
- Große Fortschritte bei statischer Sicherheitsanalyse von SW in der Zukunft
- Literatur: G. McGraw: *Software Security – Building Security In*, Addison-Wesley, 2006, www.swsec.com/